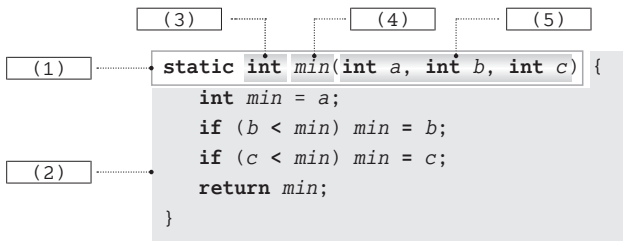
錬成問題

▪ メソッドは、プログラムの部品である。メソッドを構成する各部の名称を記入せよ。



▪ メソッドの中で宣言する局所変数の名前に関しては、以下のようになっている。

○ メソッドと同じ名前を与えることは (6) 。

○ 仮引数と同じ名前を与えることは (7) 。

▶ 共通の選択肢：(a)できる　(b)できない

▪ 上に示したメソッド *min* を呼び出す例を以下に示す。

```
min(1, 3, 2)
```

ここで使われている演算子 **()** の名称は、 (8) 演算子である。また、メソッドに対して補助的な指示を与えるための 1 や 3 や 2 は、 (9) と呼ばれる。

メソッドが呼び出されると、プログラムの流れは、そのメソッドへと移る。なお、メソッドを呼び出すことは、“メソッドを (10) する”ともいう。

呼び出されたメソッド *min* の仮引数 *a*，*b*，*c* は、 (9) の値で初期化される。このような、受け渡しのメカニズムのことを (11) と呼ぶ。

メソッド *min* は、受け取った三つの整数値の最小値を求めて、呼出し元に返す。呼出し元に返す値の型を示すのが、図中の (3) である。値を返さないメソッドの (3) は、 (12) と宣言する。なお、メソッドは二つ以上の値を一度に返すことは (13) 。

▶ (13) の選択肢：(a)できる　(b)できない

▪ メソッドによって返された値は、メソッドを呼び出した式を評価することによって得られる。*min*(1, 3, 2) を評価して得られる型は (14) で、値は (15) である。

▪ 以下に示す真理値表を埋めよ。

x	*y*	*x* & *y*
0	0	(16)
0	1	(17)
1	0	(18)
1	1	(19)

x	*y*	*x* | *y*
0	0	(20)
0	1	(21)
1	0	(22)
1	1	(23)

x	*y*	*x* ^ *y*
0	0	(24)
0	1	(25)
1	0	(26)
1	1	(27)

x	~*x*
0	(28)
1	(29)

▪ 以下に示すプログラムの実行結果を示せ。

```
System.out.println(10 & 3);
System.out.println(10 | 3);
System.out.println(10 ^ 3);
```

(30)

```
System.out.println(37 << 2);
System.out.println(37 >> 2);
System.out.println(37 >> 2);
```

(31)

▪ 以下に示すのは、受け取った二つの **int** 型の値の和を返すメソッドである（これ以降、メソッドの宣言に付けるべき **static** は省略する）。

```
int sumOf(int a, int b) {
   return  (32) ;
}
```

▪ 以下に示すのは、受け取った二つの **int** 型の値の差を返すメソッドである。

```
int diffOf(int a, int b) {
   return  (33)  ? a - b :  (34) ;
}
```

```
int diffOf(int a, int b) {
   if ( (33) )
      (35) ;
   (36) ;
}
```

▪ 以下に示すのは、受け取った二つの **int** 型の値の大きいほうの値を返すメソッドである。

```
int max(int a, int b) {
   return  (37)  ? a :  (38) ;
}
```

▪ 以下に示すのは、受け取った二つの **int** 型の値の平均を **double** 型の実数値で返すメソッドである。

```
double aveOf(int a, int b) {
   return  (39) (a + b) / 2;
}
```

▪ 以下に示すのは、受け取った二つの **int** 型の値が等しければ **true** を、そうでなければ **false** を返すメソッドである。

```
boolean equal(int a, int b) {
   return  (40) ;
}
```

▪ メソッド本体の中で宣言する変数を［ (41) ］と呼ぶのに対し、メソッドの外で宣言する変数を［ (42) ］と呼ぶ。［ (42) ］は、それが宣言されているプログラム（クラス）内のすべてのメソッドに［ (43) ］。

▶ ［ (43) ］の選択肢：(a)通用する　(b)通用しない

▪ プログラム（クラス）内で同一の名前をもったメソッドを宣言することを［ (44) ］と呼ぶ。［ (44) ］を行うためには、各メソッドの［ (45) ］が異なっている必要がある。

［ (45) ］に含まれるものには○を、含まれないものには × を示せ。

○ メソッド名 … ［ (46) ］　○ 仮引数の名前 … ［ (47) ］

○ 返却型 … ［ (48) ］　○ 仮引数の型と個数の組み合わせ … ［ (49) ］

▪ 以下に示すのは、*x* の *n* 乗を返却するメソッドである。

```
double power(double x, int n) {
   double tmp = ( (50) );
   while ( (51) > 0)
      tmp *= x;
   return tmp;
}
```

```
double power(double x, int n) {
   double tmp = ( (50) );
   for (int i = ( (52) ); i < n; i++)
      tmp *= x;
   return tmp;
}
```

▪ 以下に示すのは、文字 *c* を *n* 個連続して表示するメソッドである。

```
void putChars(char c, int n) {
   while ( (53) > 0)
      System.out.print( (54) );
}
```

▪ 以下に示すのは、前問のメソッド *putChars* を利用して、文字 '+' を *n* 個連続して表示するメソッドである。

```
void putPlus(int n) {
   ( (55) );
}
```

▪ 以下に示すのは、*month* に受け取った値に応じて、月の名前を表示するメソッドである。1，2，…，12 が、それぞれ "睦月"，"如月"，…，"師走" に対応する。なお、1 〜 12 でない値を受け取った場合は、何も表示しない。

```
void putLunarMonth(int month) {
   ( (56) )[] ms = {
      "睦月", "如月", "弥生", "卯月", "皐月", "水無月",
      "文月", "葉月", "長月", "神無月", "霜月", "師走",
   };
   if ( (57) )
      System.out.print( (58) );
}
```

▪ 以下に示すのは、受け取った **int** 型を構成する 32 ビットを、0 と 1 の並びとして表示するメソッドである（最後に改行文字を出力する）。

```
void printBitsLn(int x) {
   for (int i = [ (59) ]; i >= 0; i--)
      System.out.print(((( (60) ] & 1) == 1) ? '1' : '0');
   System.out.println();
}
```

▪ 前問のメソッド *printBitsLn* を利用する、以下に示すプログラムの実行結果を示せ。

```
printBitsLn(0);
printBitsLn(1);
printBitsLn(~0);
printBitsLn(~1);
printBitsLn(170 & 240);
printBitsLn(170 | 240);
printBitsLn(170 ^ 240);
for (int i = 0; i < 3; i++) {
   printBitsLn(120 << i);
   printBitsLn(120 >> i);
}
printBitsLn(~0 << 5);
printBitsLn(~0 >> 5);
printBitsLn(~0 >>> 5);
```

(61)

▪ 以下に示すのは、受け取った **int** 型を構成する 32 ビットのうち 0 であるビットの個数を求めて表示するメソッドである。

```
int count0Bits(int x) {
   int bits = 0;
   for (int i = 0; i < [ (62) ]; i++) {
      if ( [ (63) ] ) bits++;        // xの最下位ビットは1であるか？
      x [ (64) ] 1;                  // 調べたxの最下位ビットを弾き出す
   }
   [ (65) ] bits;
}
```

▪ 以下に示すのは、受け取った **int** 型の配列の要素の値の範囲（最大値と最小値の差）を求めて返却するメソッドである。

```
int rangeOf( [ (66) ] a) {
   int min = a[ (67) ];
   int max = a[ (68) ];
   for (int i = [ (69) ]; i < [ (70) ]; i++) {
      if (a[i] < [ (71) ])
         min = [ (72) ];
      if (a[i] > [ (73) ])
         max = [ (74) ];
   }
   return [ (75) ];
}
```

▪ 右に示すのは、受け取った **int** 型の配列の最大値をもつ要素のインデックスを求めて返却するメソッドである。最大値をもつ要素が複数存在する場合は、最も先頭のインデックスを返却する。

たとえば、配列の要素が {1, 3, 5, 4, 5, 3} であれば、最大値 5 をもつ要素の最も先頭要素のインデックスである 2 を返却する。

```
int maxIndexOf(int[] a) {
   int max = a[0];
   int idx = [ (76) ];

   for (int i = 1; i < a.length; i++) {
      if (a[i] > [ (77) ]) {
         max = [ (78) ];
         idx = [ (79) ];
      }
   }
   [ (80) ];
}
```

▪ 以下に示すのは、**int** 型配列 *b* の全要素を **int** 型配列 *a* にコピーする（同一インデックスの要素に値をコピーする）メソッドである。なお、配列の要素数が異なる場合は、小さいほうの要素数分だけコピーするものとする。

```
void copyArray(int[] a, int[] b) {
   int n = a.length < b.length ? [ (81) ] : [ (82) ];
   for (int i = 0; i < n; i++)
      a[i] = [ (83) ];
}
```

▪ 以下に示すのは、**int** 型配列 *a* の全要素に *k* を代入するメソッドである。

```
void arrayFill([ (84) ] a, int k) {
   for (int i = 0; i < a.length; i++)
      a[i] = [ (85) ];
}
```

▪ 以下に示すのは、要素数が *n* で全要素の値が *k* である **int** 型配列を生成し、その配列（への参照）を返却するメソッドである。

```
[ (86) ] arrayFillOf(int n, int k) {
   int[] a = [ (87) ];
   for (int i = 0; i < n; [ (88) ])
      [ (89) ] = k;
   return [ (90) ];
}
```

▪ 以下に示すのは、**int** 型配列 *a* の末尾に *b* をつなげた配列を生成し、その配列（への参照）を返却するメソッドである。たとえば、配列 *a* が {1, 2, 3} で配列 *b* が {2, 4, 6, 8} であれば、返却する配列は {1, 2, 3, 2, 4, 6, 8} である。

```
[ (91) ] arrayCatOf([ (92) ] a, [ (93) ] b) {
   int[] z = [ (94) ];
   for (int i = 0; i < [ (95) ]; i++) z[ [ (96) ] ] = a[i];
   for (int i = 0; i < [ (97) ]; i++) z[ [ (98) ] ] = b[i];
   return [ (99) ];
}
```

▪ 以下に示すのは、受け取った **int** 型の配列 *a* に基づいて **int** 型の 2 次元配列を生成して、その配列（への参照）を返却するメソッドである。生成する配列の行数は *a*.length で、

第*i*行の列数は*a*[*i*]として、全要素の値は0とする。たとえば、配列*a*が{3, 2, 4}であれば、返却する配列は、{{0, 0, 0}, {0, 0}, {0, 0, 0, 0}}である。

```
  (100)  array2DOf(  (101)  a) {
    (102)  c = new  (103) ;
    for (int i = 0; i <  (104) ; i++)
      c[  (105) ] = new  (106) ;
    (107)  c;
}
```

▪ 以下に示すのは、受け取った**int**型の２次元配列*a*の複製を生成して、その配列（への参照）を返却するメソッドである。

```
  (108)  array2Dclone(  (109)  a) {
    (110)  c = new  (111) ;
    for (int i = 0; i <  (112) ; i++) {
      c[  (113) ] = new  (114) ;
      for (int j = 0; j <  (115) ; j++)
        c[i][j] =  (116) ;
    }
    return  (117) ;
}
```

▪ 以下に示すのは、２次元配列*x*と*y*に格納された行列の和を生成して、その配列（への参照）を返却するメソッドである。なお、行数や列数が異なる場合は、行数・各行の列数は要素数の大きいほうにあわせ、存在する側の要素の値をそのまま格納する。

```
  (118)  addMatrix(  (119)  x,  (120)  y) {
    int heightMin, heightMax, maxH;
    if (  (121)  >  (122) ) {
      heightMin = y.length;  heightMax = x.length;  maxH = 0;
    } else {
      heightMin = x.length;  heightMax = y.length;  maxH = 1;
    }
    (123)  c = new int[heightMax][];
    int i = 0;
    for ( ; i < heightMin; i++) {
      int widthMin, widthMax, maxW;
      if (  (124)  >  (125) ) {
        widthMin = y[i].length;  widthMax = x[i].length;  maxW = 0;
      } else {
        widthMin = x[i].length;  widthMax = y[i].length;  maxW = 1;
      }
      c[i] = new int[  (126) ];
      int j = 0;
      for ( ; j <  (127) ; j++) c[i][j] = x[i][j] + y[i][j];
      if (maxW == 0)
        for ( ; j <  (128) ; j++) c[i][j] =  (129) ;
      else
        for ( ; j <  (130) ; j++) c[i][j] =  (131) ;
    }
    if (maxH == 0) {
      for ( ; i <  (132) ; i++) {
        c[i] = new int[  (133) ];
        for (int j = 0; j <  (134) ; j++) c[i][j] =  (135) ;
      }
    } else {
      for ( ; i <  (136) ; i++) {
        c[i] = new int[  (137) ];
        for (int j = 0; j <  (138) ; j++) c[i][j] =  (139) ;
      }
    }
    return c;
}
```